Photon Escape Simulation from the Sun's Core

Objective

Simulate the time it takes for a photon to escape from the Sun's core to its surface using a 1D random walk model.

Assumptions

1. The Sun is a perfect sphere.
2. The photon's movement can be modeled as a 1D random walk. It has some probability to move right (towards the radius edge) and left (towards the centre).
3. Dominant pressures that bias the photon to move right are thermal, radiative, and gravitational.
4. These can be modelled as a net 'force' that acts on the photon and so P(right) > P(left).
5. The bias probability P represents the net outward 'force' on the photon.
6. We assume the interaction time is negligible.

Constants

In [ ]:
```
# Solar constants
R = 6.96e8  # Radius of the Sun in meters
lambda_mean = 1e-3  # Mean free path in meters

# Constants for density
density_core = 1.62e5  # kg/m^3, density at the core
density_surface = 2.0e-4  # kg/m^3, density at the surface

c = 3e8  # Speed of light in m/s
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import time
import pandas as pd
```

In [ ]:
```
# calculate how many steps if the photon always moves right
R/lambda_mean  # 696000000000 steps
```

Out[ ]:
```
696000000000.0
```

Here's our code for simulating the time it takes for a photon to leave the sun

In [ ]:
```python
# Function to simulate photon escape time and return positions and steps in
def simulate_photon_escape_time(bias_prob, radius=int(R), max_steps=int(1e3))
    position = 0  # Start at the center of the Sun
    time_taken = 0  # Time taken for photon to escape
    current_steps = 0  # Start counting how many steps we've taken
    data = {'Steps': [], 'Position': []}  # For storing data

    while position < radius:
        # Generate a random number to decide the direction
        rand_num = np.random.rand()

        # Apply bias probability
        if rand_num <= bias_prob/2 + 0.5:
            position += lambda_mean  # Move towards the surface
        else:
            position -= lambda_mean  # Move towards the core

        # Increment the steps
        current_steps += 1

        # Store data
        data['Steps'].append(current_steps)
        data['Position'].append(position)

        # Check if photon has escaped
        if current_steps >= max_steps:
            print(f'simulation ended prematurely after {current_steps} steps
            print(f'photon travelled {position:.7f}m from the centre of the
            break

    # Create DataFrame
    df = pd.DataFrame(data)
    return df

# Function to plot photon path
def plot_photon_path(df):
    plt.figure(figsize=(10, 6))
    plt.plot(df['Steps'], df['Position'], label='Photon Path', color='b')
    plt.xlabel('Steps')
    plt.ylabel('Position (m)')
    plt.title('Photon Path')
    plt.grid(True)
    plt.legend()
    plt.show()

# Run the simulation and plot
bias_probability = 7.353e-14  # decimal between 0 and 1. 0 is no bias, 1 is
df = simulate_photon_escape_time(bias_probability, max_steps=1e7)
plot_photon_path(df)
```

```
simulation ended prematurely after 10000000 steps
photon travelled 2.2360000m from the centre of the sun, 0.00000032% of the w
ay to the surface of the sun
```

Ok so this shows our simulation works but it doesn't travel very far! This is likely due to the sheer number of collisions that occur even though the photons are moving at *c*, and we're trying to simulate every interaction

Let's see how long it would take to simulate all the way to the edge of the sun

In [ ]:
```
int(df['Steps'].iloc[-1])
```

Out[ ]:
```
10000000
```

In [ ]:
```
start_time = time.time()
df = simulate_photon_escape_time(bias_probability, max_steps=1e7)
num_steps = int(df['Steps'].iloc[-1])
elapsed_time = time.time() - start_time
print(f'num_steps: {num_steps:.10f}, calculation_time: {elapsed_time:.6f}s')
```

```
simulation ended prematurely after 10000000 steps
photon travelled -1.5700000m from the centre of the sun, -0.00000023% of the
way to the surface of the sun
num_steps: 10000000.0000000000, calculation_time: 7.363748s
```

In [ ]:
```
percent_to_edge = (100 * df['Position'].iloc[-1]) / R
print(f'percent to edge of sun: {percent_to_edge:.10f}%')

time_to_compute_to_escape = (elapsed_time) / (percent_to_edge / 100)

print(f'time to complete simulation: {time_to_compute_to_escape/3600:.5f} ho
```

```
percent to edge of sun: -0.0000002256%
time to complete simulation: -906788.45140 hours
```

It would take ~700 hours to complete every interaction! So we'll have to estimate and extrapolate the time it takes to exit the sun with this simulation

In [ ]:
```
# calculate the time per step (interaction)
time_per_step = lambda_mean/c
print(time_per_step)
```

```
3.3333333333333335e-12
```

In [ ]:
```
# extrapolate how many seconds it would take to escape

# how many seconds have elapsed
seconds_elapsed = time_per_step * num_steps

time_to_escape = (seconds_elapsed) / (percent_to_edge / 100)

print(time_to_escape)  # in seconds
```

```
-14777.070063694851
```

In [ ]:
```
# extrapolate how many seconds it would take to exit
time_per_step * (1/ ((percent_to_edge / 100) /num_steps))

# how many seconds have elapsed
seconds_elapsed = time_per_step * num_steps

# how far through are we
percent_to_edge / 100

seconds_elapsed/percent_to_edge

time = (100/(percent_to_edge/num_steps)) * time_per_step
print(f'{time/(3600*24*365)}')
```

```
-0.0004685778178492787
```

NEXT STEPS

- The above assumes a constant density, which is a poor assumption. Try a range of values of the bias probability to see what the net force effect is actually like (how often does it actually seem to bias moving left over right)
- Experiment with different density distributions

In [ ]:
```

```

In [ ]:
```
def simulate_photon_escape_time_with_density(bias_prob, steps=int(1e6)):
    position = 0  # Start at the center of the Sun
    time_taken = 0  # Time taken for photon to escape

    for _ in range(steps):
        # Generate a random number to decide the direction
        rand_num = np.random.rand()

        # Calculate current density
        current_density = density_core + density_slope * position

        # Calculate mean free path based on current density
        lambda_current = lambda_mean * (density_core / current_density)

        # Apply bias probability
        if rand_num < bias_prob:
            position += lambda_current  # Move towards the surface
        else:
            position -= lambda_current  # Move towards the core

        # Calculate time taken for this step
        time_step = lambda_current / c
        time_taken += time_step

        # Check if photon has escaped
        if position >= R:
            break
    return time_taken  # Time taken in seconds
```

Density Modelling

The above assumes constant density of the sun which is a poor assumption. Let's experiment with different density profiles

In [ ]:
```
# Create an array of positions from the center to the surface of the Sun
positions = np.linspace(0, R, 1000)

density_slope = (density_surface - density_core) / R  # Linear rate of chang
# Calculate the density at each position
densities = density_core + density_slope * positions

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(positions, densities, label='Density Profile')
plt.xlabel('Position (m)')
plt.ylabel('Density (kg/m^3)')
plt.title('Density Profile of the Sun from Core to Surface')
plt.legend()
plt.grid(True)
plt.show()
```

In [ ]:
```
# Constants for exponential density
decay_constant = -np.log(density_surface / density_core) / R

# Plotting the exponential density profile
exp_densities = density_core * np.exp(-decay_constant * positions)

plt.figure(figsize=(10, 6))
plt.plot(positions, exp_densities, label='Exponential Density Profile')
plt.xlabel('Position (m)')
plt.ylabel('Density (kg/m^3)')
plt.title('Exponential Density Profile of the Sun from Core to Surface')
plt.legend()
plt.grid(True)
plt.show()
```

This looks better but decays quite steeply. The reality is more likely to be less dramatic than this

In [ ]:
```
# Adjust decay constant based on the half-life
half_life = 1e8  # Half-life in meters
decay_constant_adjusted = np.log(2) / half_life

# Plotting the adjusted exponential density profile
adjusted_exp_densities = density_core * np.exp(-decay_constant_adjusted * po
plt.figure(figsize=(10, 6))
plt.plot(positions, adjusted_exp_densities, label='Adjusted Exponential Dens
plt.xlabel('Position (m)')
plt.ylabel('Density (kg/m^3)')
plt.title('Adjusted Exponential Density Profile of the Sun from Core to Surf
plt.legend()
plt.grid(True)
plt.show()
```

This looks like it might be more representative. We can validate this at a later point.

In [ ]:
```
import matplotlib.pyplot as plt
import time
import numpy as np

# Constants
R = 6.96e8  # Radius of the Sun in meters
c = 3e8  # Speed of light in m/s
lambda_mean = 1e-2  # Mean free path in m
decay_constant_adjusted = np.log(2) / 3.5e8  # Adjusted decay constant
density_core = 1.62e5  # kg/m^3, density at the core

def simulate_photon_escape_time(bias_prob, radius=int(R), max_steps=int(1e2)
    position = 0  # Start at the center of the Sun
    time_taken = 0  # Time taken for photon to escape
    current_steps = 0  # Start counting how many steps we've taken
    positions = []  # For visualization

    while position < radius:
        # Generate a random number to decide the direction
        rand_num = np.random.rand()

        # Apply bias probability
        if rand_num < bias_prob/2 + 0.5:
            position += lambda_mean  # Move towards the surface
        else:
            position -= lambda_mean  # Move towards the core

        # Calculate time taken for this step
        time_step = lambda_mean / c
        time_taken += time_step

        # Increment the steps
        current_steps += 1

        # For visualization
        if visualize:
            positions.append(position)

        # Check if photon has escaped
        if current_steps >= max_steps:
            print(f'simulation ended prematurely after {current_steps} steps
            print(f'photon travelled {position:.5f}m from the centre of the
            break

    if visualize:
        # Plotting Photon Path
        plt.figure(figsize=(10, 6))
        plt.plot(positions, label='Photon Path')
        plt.xlabel('Steps')
        plt.ylabel('Position (m)')
        plt.title('Photon Path from Core to Surface')
        plt.legend()
        plt.grid(True)
        plt.show()

        # Plotting Density Profile Overlay
        exp_densities = density_core * np.exp(-decay_constant_adjusted * np.
        plt.figure(figsize=(10, 6))
        plt.plot(positions, exp_densities, label='Exponential Density Profil
        plt.xlabel('Position (m)')
        plt.ylabel('Density (kg/m^3)')
        plt.title('Density Profile Along Photon Path')
        plt.legend()
        plt.grid(True)
        plt.show()

    return time_taken  # Time taken in seconds

# Run the simulation with visualization
bias_probability = 1.5e-13  # decimal between 0 and 1. 0 is no bias, 1 is al
start_time = time.time()
escape_time = simulate_photon_escape_time(bias_probability, visualize=True)
elapsed_time = time.time() - start_time
print(f'escape_time: {escape_time:.10f}s, calculation_time: {elapsed_time:.5
```

```
simulation ended prematurely after 100 steps
photon travelled -0.14000m from the centre of the sun, -0.000000% of the way
to the surface of the sun
```

```
escape_time: 0.0000000033s, calculation_time: 0.46339s
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
simulation ended prematurely after 1000 steps
photon travelled 2.20000m from the centre of the sun, 0.000000% of the way t
o the surface of the sun
```

```
In [ ]:
```